AD-HOC HID: Modular Wireless Human Interface Device

Joe Mazzanti
Electrical Engineering
8 June 2014

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Dr. Bridget Benson, for advising this project. I would also like to thank Dr. Kevin Taylor of the Kinesiology department and Dr. Lily Laiho of the Biomedical Engineering department. Both professors were useful consultants regarding the marketing and manufacturing of a prototype device. I would lastly like to thank Garrett Gudgel, Michael Schier, and Tim Grijalva, who helped me to actually finish the prototype device. They helped me machine housings, optimize code, and use Mechanical Engineering resources that are normally unavailable to Electrical Engineers.

Abstract

Ad-Hoc HID is a modular, reprogrammable Human Interface Device. This device is intended to function as a keyboard, gamepad, or mouse, according to the user's needs. The final project is intended to be switch agnostic, making the final product adaptable to the user's needs.

Introduction

"Ad-Hoc," is a Latin phrase that can roughly translate to "For this purpose." The Bluetooth protocol also supports creating short-range Ad-Hoc networks in order to communicate with hardware devices. This name seemed to perfectly define the ultimate objectives of this project.

Human interface devices are a class of computer accessories that interact directly with people, such as keyboards, joysticks, gamepads, and mice. These interfaces are meant to translate human movement into a computer input. These devices are universally required in order to use a computer. However, due to certain handicaps, not every human is able to utilize every human interface device.

Mass produced handicap accessible computer interface devices are expensive. Individual custom computer hardware is even more expensive. A simple gamepad with a digital joystick and two digital buttons can cost about $250. A handicap accessible pushbutton can cost $100. A more complex gamepad can cost more than $1,000. Medical insurance companies will often ignore these exorbitantly high costs, because game pads are not considered "necessary."

Handicap accessible computer interface devices comprise a considerably large product category. However, the range of disabilities is increasingly larger than the range of available products. Many types of adaptive technologies exist today. Specialized switches are available through online retailers; eye-tracking and voice-command software is becoming increasingly more available.

The original inspiration for this project came from a controller made by Caleb Kraft. Caleb made his controller to help a boy with muscular dystrophy play computer games. This project expands on his idea by producing a platform that can adapt to other types of disabilities. Ideally, this platform can be used to produce unique controls to adapt to unique people, instead of having to adapt a unique person to a mass-produced controller. While the prototype from this project can be seen mainly as

another muscular dystrophy solution, the real benefit of this platform is its adaptability through its modularity and expandability.

The entire project will be made with the hope that custom; handicap accessible human interface devices can be made cheaply, without requiring advanced technical knowledge. The entire project was designed and assembled with readily available parts. The final product proves that custom computer accessories can be made on a $90 budget.

Requirements

The system requirements are outlined as follows:

- Reprogrammable
- Wireless with at least 10 ft. operating range
- Less than $150 parts cost
- Available to produce using readily available tools and equipment
- Battery Powered
- Modular
- Mac and Windows Compatible
- Device automatically detected as wireless keyboard.
- Future Expandability
- Less than 16 ms. latency between controller sending a command and the operating system receiving it.

The main goal for this system is to be user expandable. This means that the system is mainly designed to be flexible. Flexibility comes from sticking to a basic set of requirements, and making sure those requirements are perfectly satisfied, while allowing room to add new requirements as they are needed. Each system specification is analyzed in the next section of the report.

Specifications

Due to the nature of this project, the technical specifications are the same as the marketing requirements. There is nothing this system can do that only an engineer should understand. Instead, the goal is to have a system that can be built and used by anyone, even if they are not an engineer.

Here is a cost breakdown for the project, including a complete inventory. The project cannot be completed without these parts or something equivalent. However, some possible, cheaper substitutions can be made.

Table 1 - Project Costs and Inventory

Part	Description	Quantity	Cost	Total
Arduino Pro Mini 328	5V, 16 MHz	1	$9.95	$9.95
BlueSMiRF HID	Bluetooth Modem	1	$44.95	$44.95
Breakaway Headers	Set of Metal Headers	1	$2.95	$2.95
Plastic Enclosure	5.50 x 4.25 x 1.75 in.	2	$4.34	$8.68
Bag of Enclosure Screws	#4-24 x 1/4" Philips	1	$3.50	$3.50
30 mm Pushbutton	Sanwa OBSF-30	6	$2.75	$16.50
24 mm Pushbutton	Sanwa OBSF-24	2	$2.25	$4.50
Leaded Solder	60/40, 0.8 mm	1	$9.95	$9.95
8 way directional joystick	Sanwa JLF	1	$23.95	$23.95
Stranded Wire	24 AWG	1	$5.97	$5.97
Soldering Iron	40 W	1	$42.39	$42.39
9V Battery		1	$2.62	$2.62
AVR Programmer	ISP for AVR	1	$14.95	$14.95
9v Battery Snap Connector		1	$1.25	$1.25
Total Cost				$192.11

The total cost for my prototype was $190.86. I originally stated that the cost of the project could be about $100. I say this because substituting the parts I chose for cheaper versions can reduce many of the expenses. For instance, a functional Soldering Iron can be purchased for about $10.00. A cheaper joystick can be purchased for $15.00 and the pushbuttons can be purchased for $2.00 each. The enclosure screws can be purchased separately from a hardware store. Buying just 8 screws should be much cheaper than buying a bag of 100.

An Arduino Uno can be used instead of an Arduino Pro Mini, however this doubles the size and cost of the microcontroller. This may be a better option though, as many people already have an Arduino Uno, and if you use the Uno, you also reap two distinct benefits: You can use USB power for the system, and you do not need to purchase the AVR Programmer.

If you already have the Arduino Uno and insist on using the Pro Mini instead, you can use the Arduino Uno as an AVR Programmer, and do not need to purchase the AVR programmer separately. I programmed the whole project using the Arduino IDE, which can be installed from

http://arduino.cc/en/main/software.

Design

The system hardware and software are designed with expandability in mind. For this to work, the basic system functions must perform near flawlessly. My design considerations will be explicitly stated because some choices require explaining. If there is more than one way to perform a task, I will explain what I did, and what I learned in the process.

Software Design

Initial Setup

The Bluetooth modem used for the project is a Roving Networks RN-42 HID. This was chosen for ease of programming. The module expects a specially formatted packet to be sent. If the packet is sent properly, the Bluetooth modem properly encodes and transmits the HID report without requiring any other interaction. This is convenient, as it allows anyone to create Bluetooth HID devices with very little experience, and no custom firmware needs to be written.

The Bluetooth modem is sent from the retailer in “command mode.” Command mode allows the user to configure parameters such as the baud rate, device password, or a remote address for the module to automatically connect to upon initializing. These can be set using a terminal emulator, or directly in software using the microcontroller.

The manufacturer has a produced a data sheet [1] that explains how to enter command mode on the Bluetooth modem. This data sheet also has a complete list of commands for the Bluetooth modem.

The main instructions to enter command mode are available in the manufacturer’s datasheet. I used command mode to change the Baud Rate and the device mode.

The most important commands to set from a terminal program are the following:

SZ,1024

This command sets the Baud Rate of the Bluetooth modem to 250000. With a 16 MHz microcontroller clock, this baud rate gives a 0% clock error.

This is ideal, because it minimizes any potential timing errors.

It is also necessary to set the HID profile in the Bluetooth modem's firmware. If this command is not sent, then the device will not be detected as an HID device. To set the module as an HID device, follow the manufacturer's datasheet to enter command mode. From here, send the following characters:

S~,6

This command will let a computer detect the Bluetooth HID device and properly decode its transmissions as HID Reports.

These commands are the two most important commands, and should be done through a terminal program. My project only requires this two. Any other functions detailed in the RN-42 datasheet can be configured during the microcontroller's initialization sequence.

Sending HID Reports

HID Reports are specially formatted strings that are of a specified length. These reports are sent to the Bluetooth modem, which then interprets them into key press and key release commands. This Bluetooth module only allows six simultaneous key presses plus any combination of modifiers, such as Shift, Ctrl. or Alt.

Here is an example of a properly formatted raw HID report.

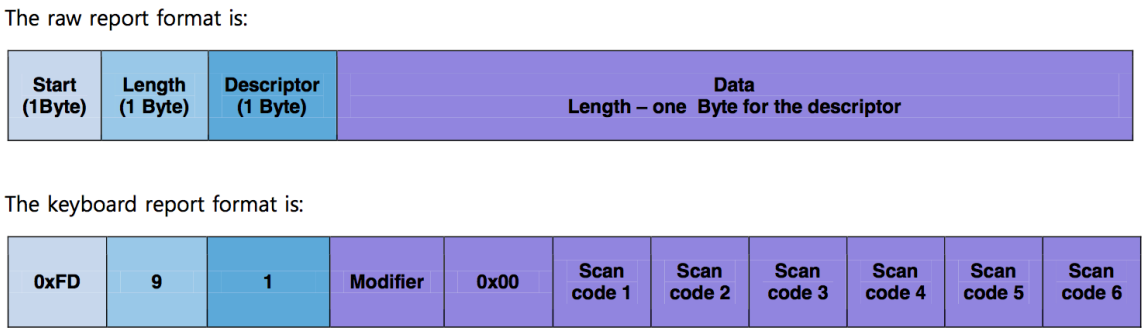

Figure 1 - HID Raw Report

In code, this is constructed and then sent serially, one byte at a time. Once a button press signal is received by the operating system, the operating system will remember which buttons are currently being pressed down. A release signal must be sent for the operating system to release the button.

Typically, it is best practice to release a button before sending a different button. It is possible to send multiple buttons at once without releasing other buttons, but this may lead to releasing certain keys prematurely.

For example, imagine a case where six buttons are being held down, call them Buttons 1 – 6. An HID report will be sent with Button 1 sent as “Scan Code 1” according to Figure 1. Buttons 2-6 will be sent as “Scan Codes 2-6.”

Now, without releasing Buttons 1-6, Button 7 is pressed. There are two options:

1. Since there is no “Scan Code 7” spot, a spot from “Scan Code 1-6” needs to be used to transmit Button 7’s press.

   Or:

2. Button 7 will not be transmitted until any other pressed Button is released. Once a Button is released, Button 7 will occupy its position in “Scan Code 1-6.”

I chose to use option 2, mainly because it is so difficult to actually press 6 buttons simultaneously. The issue with option 1 is that the operating system is aware of which Scan Code slot is used to transmit a key press. Using the “Scan Code 1” slot to transmit Button 1 and Button 7 is complicated, because sending the value “0x00” in the “Scan Code 1” slot will release every button whose key press command has been sent in that slot, even if one of the buttons is still being held down. This means the button would need to be physically released and then pressed again in order for the operating system to receive another button press command.

Flow Diagram and Algorithm Decomposition.

The code algorithm works according to the following outline. Steps 1-4 can be seen in the Flow Diagram of Figure 2. Steps 5-9 can are shown in Figure 3.

1. Initialize the microcontroller, initialize current joystick position and buttons pressed variables to 0.
2. The 50k**Ω** pull-up resistors of the inputs are enabled and the microcontroller’s GPIO pins are set to the proper data direction.
3. Set the baud rate of the microcontroller to 250000 and start a serial connection to the Bluetooth Modem. (At this point, the Bluetooth

modem can be connected to from a computer with Bluetooth capabilities.)

4. Read the state of the joystick and the buttons by reading the GPIO pins. Store the initial state as a variable

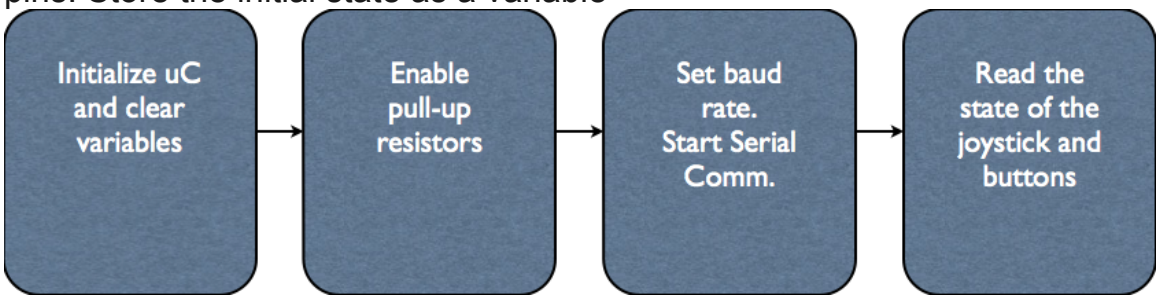

Figure 2 - Initialization Sequence Flow Diagram

5. Enter a loop that polls for the joystick and button states to change. To detect this, constantly read the joystick and button pins and compare their value to the joystick and button variable.
6. If the joystick or the button states do not match the variable's value, then a change has occurred, and a new HID report must be constructed and sent to the operating system
7. The new joystick and button states are detected and loaded into the HID report. A maximum of six commands can be sent at one time. Typically, this is two joystick directions, and four buttons.
8. The new joystick and button states are assigned to the joystick and button state variable, which is then used to compare to future joystick and button states.
9. This program continues looping steps 5 – 8 until powered off.

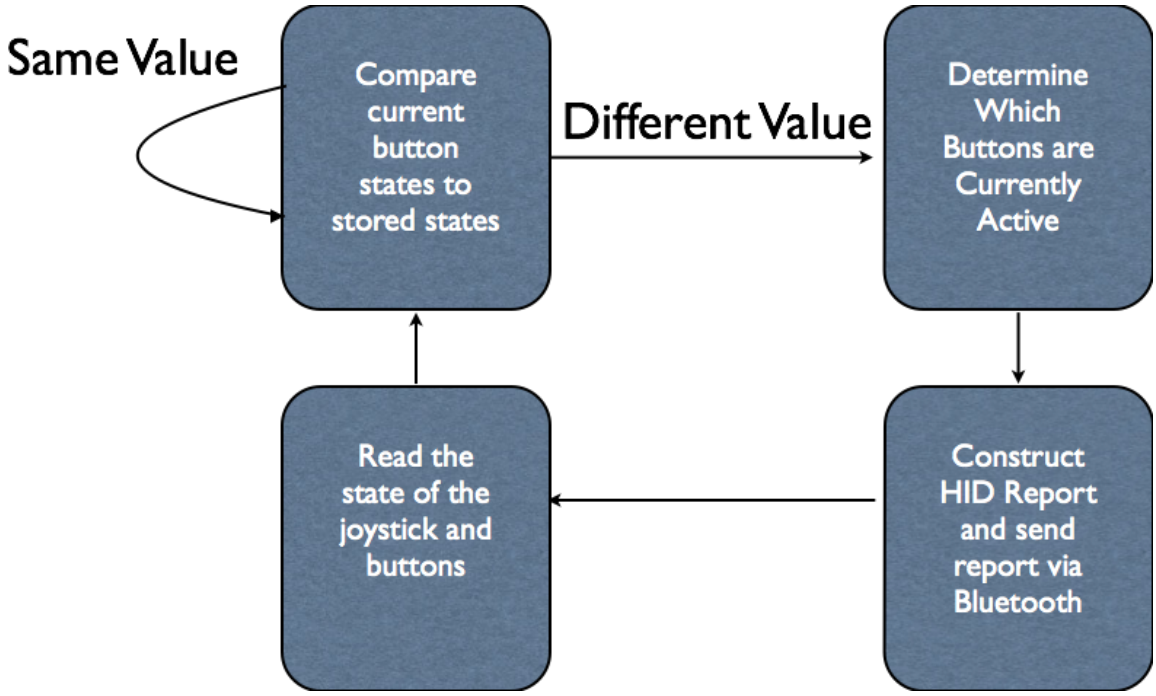

Figure 3 - Main Software Loop Flow Diagram

Function Descriptions

There are seven main functions that are used to determine which buttons are being pressed, and using this information, compose HID reports. Here is the hierarchy used for the program to collect data about the button states, and make HID reports. The functions are listed in the order they pass data to each other.

getKeyMask()
getKeyMask() is used to check the pressed buttons to an explicitly defined bit mask value. This information is used later with getPortForKeyValue() in order to tell getState() the current status of the buttons and joystick.

getPortForKeyValue()
getPortForKeyValue() is used to determine which port the change has been detected on. Since the joystick is on Port D and the buttons are on Port B, this function is used with the value from getKeyMask() in order to tell getState() the port and bit position in that port has been changed.

getState()
The getState function is used to determine the status of the joystick and buttons. The joystick and buttons are connected directly to the

microcontroller pins. When the buttons are not being pressed, the pin reads a high value due to the pull-up resistor. When the button is pressed, a connection to ground is made, and the voltage at the input pin drops to a low value.

getPortBKeyValue()
The values of Port B's switches can be determined. Each of these switches is mapped to a keyboard letter. The PortBKeyValue function performs this mapping. Each individual Port B switch directly corresponds to a unique key on the keyboard.

getPortDKeyValue()
This function is the same as getPortBKeyValue() but gets the value of the joystick instead of the pushbuttons.

sendButtons()
sendButtons() is used to create the HID report and send it to the Bluetooth module. It is called as a part of updateActiveKeys().
updateActiveKeys()
updateActiveKeys() is used to control the Scan Codes being sent in the HID Report. The function uses an array called "activeKeys" in order to keep track of six currently active keys. If no buttons are being pressed, the whole activeKeys array is assigned a value of 0x00. When a button is pressed, the function steps through the array looking for a value of 0x00. When a value of 0x00 is found in the array, the function assigns the HID hex value to that position in the array. When all six spots in the array or filled, or when every key press has been recorded, the sendButtons() function is called to transmit the HID Report.

Upon key release, the updateActiveKeys() function steps through the array, comparing the released key's value to the values stored in the array. When the released key's value is found in the array, a value of 0x00 replaces the key's value in that index position of the array. sendButtons() is called and a new HID report is transmitted to tell the operating system which keys are still being pressed, and which keys have been released.

Hardware Design

The hardware design can be detailed in two distinct phases. Phase 1 of the hardware prototype design involves configuring the microcontroller and ensuring the proper function of the entire system. Phase 2 of the design involves building a housing for the prototype, which should result in a functional product.

System Circuit Design

The Level 0 functional decomposition can be seen in Figure 2. From a hardware perspective, the system is intended to be as straightforward and reliable as possible.

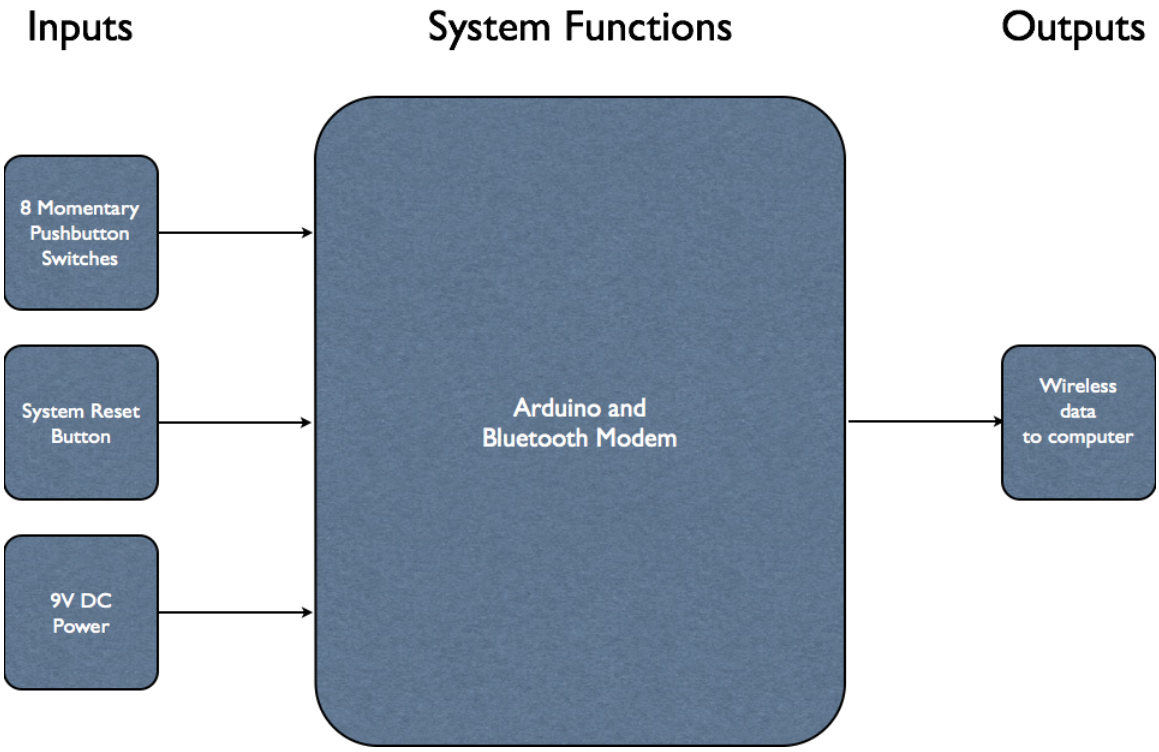

Figure 4 - Level 0 Functional Decomposition

The microcontroller is the main component of the system. Every signal in Figure 2 is connected directly to the microcontroller, and the microcontroller connects directly to the Bluetooth modem. Normally, when a button is used, a large valued resistor is used to limit the current draw of the GPIO pin, and to establish consistent logic values by keeping the pin from floating. In this design, the pull-up resistors on the microcontroller remove and need to add discrete resistors. This is beneficial, as it allows the buttons to connect directly between our input and ground. This simplifies the system wiring and reduces the size of the system.

System Wiring

The wiring between the Arduino and the RN-42 is shown in Figure 5's diagram. The Arduino Pro Mini is ideal for this, because its PCB is laid out in a way that can easily connect to the BlueSMiRF HID PCB. The Bluetooth modem only needs to receive serial transmissions, it does not need to send them back to the microcontroller. This is convenient because the entire modem can be connected with only three wires: Vcc, Gnd, and Rx. The Arduino has a voltage regulator on its Vcc pin. If a voltage

greater than 5v is tied to the voltage pins, the voltage regulator will step it down to a usable 5v.

Figure 5 shows how the Bluetooth module needs to be connected to the Arduino.

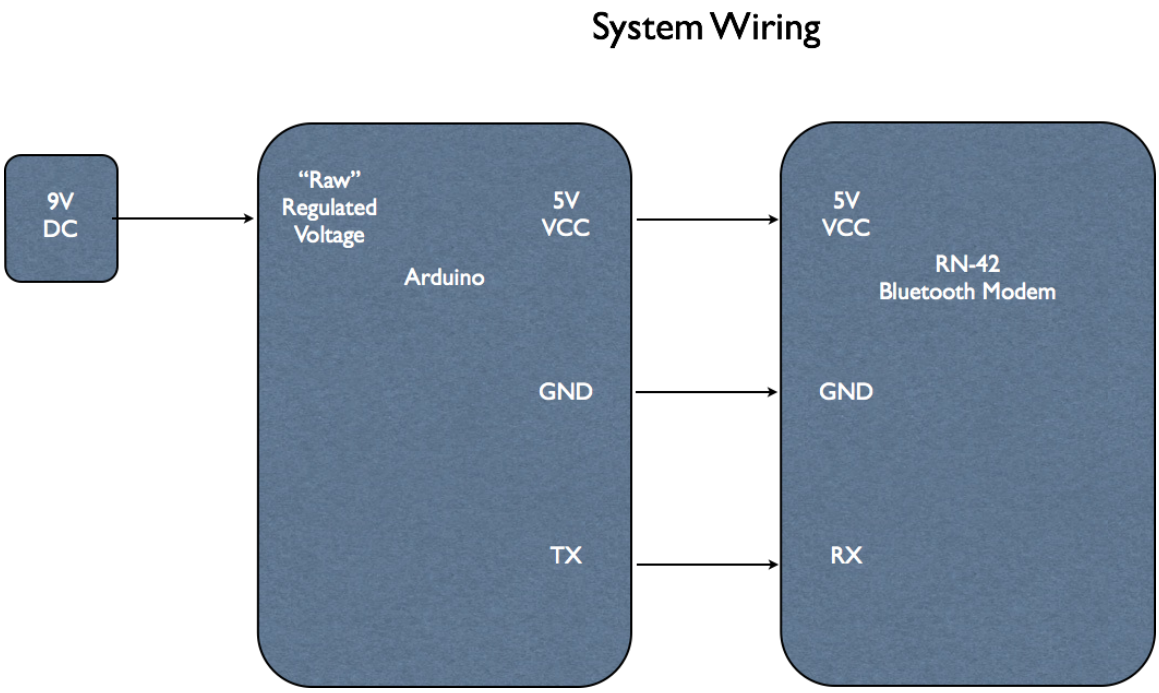

Figure 5 - Arduino to Bluetooth Module Wiring Diagram

To power the system, a 9v battery is connected to the regulated voltage input and to ground. It is wise to connect a switch in between the battery and the Arduino in order to turn the system on and off at will. Instead of soldering the battery directly to the Arduino, use a 9v battery snap connector so the battery can be easily swapped.

Avoid soldering the buttons and joystick to the Arduino until they are mounted in the housing. Otherwise, it will be nearly impossible to mount these components without desoldering.

Button Wiring

The button wiring is intended to save space while maintaining functionality. Figure 6 shows the button-wiring diagram for the system. Each pushbutton has two pins. When the button is pressed, the switches are shorted together, and current is allowed to flow across the switch. Knowing this, it is possible to tie one pin of each button together, so the form one large node. This node is then connected to a common ground on the microcontroller. Each button has its other pin tied to a distinct pin on the microcontroller. This scheme is effective for reducing the total

length of wire used, as only one ground wire needs to be connected to the microcontroller. The final wiring can be seen in Figure 12.

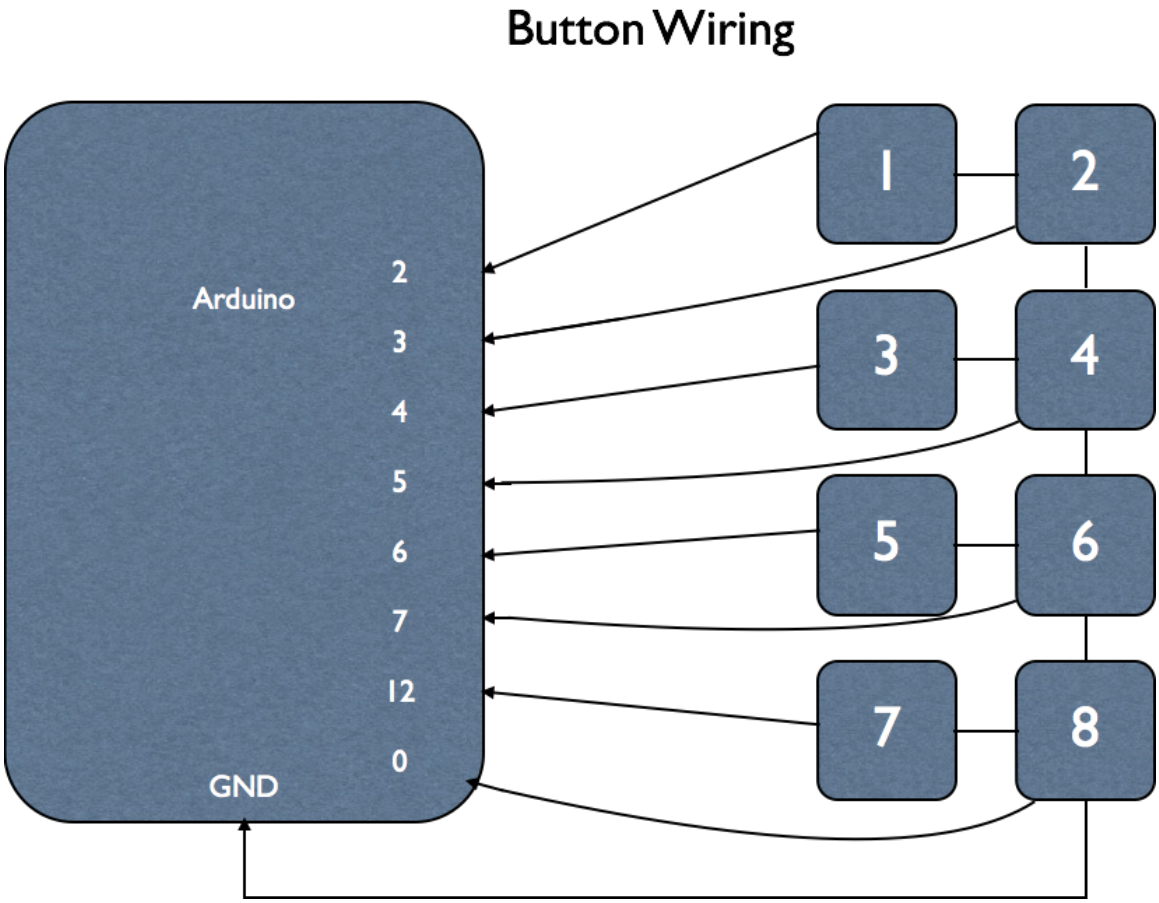

Figure 6 - Button Wiring Diagram

Joystick Wiring

The joystick controls direction in games by emulating the W,A,S, and D keys of a keyboard. These keys are typically chosen to control character movement in games.

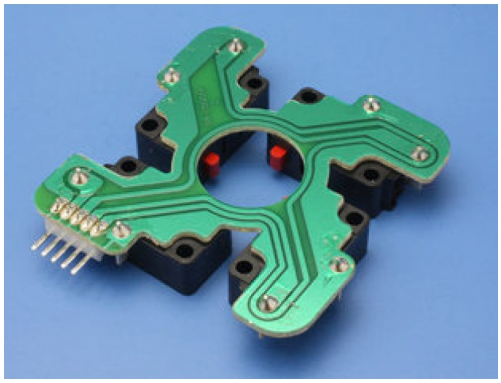

Figure 7 - Joystick PCB

The joystick is a set of four switches, which share a common ground. Figure 7 shows what a typical joystick PCB looks like. The joystick has a 5-pin connector, which is used to connect the joystick to a PCB. Figure 8 shows a wiring harness that is made to connect to the joystick PCB. Due the to plastic lip on the wiring harness, it can only be attached one way.

Figure 8 - Joystick wiring harness

Because the harness can only be connected one way, it is easier to reference the wiring diagram based on the physical location of each wire. Wire color cannot be

used as a reference because different wiring harnesses potentially have different colored wires. Since the harness just holds the wires in place, position is the most common method of identifying the proper layout.

The joystick wiring diagram in Figure 9 is organized as if the wiring harness is properly connected, and the joystick is being viewed on the side of the PCB and restrictor gate, instead of the side of the ball top and dust washer.

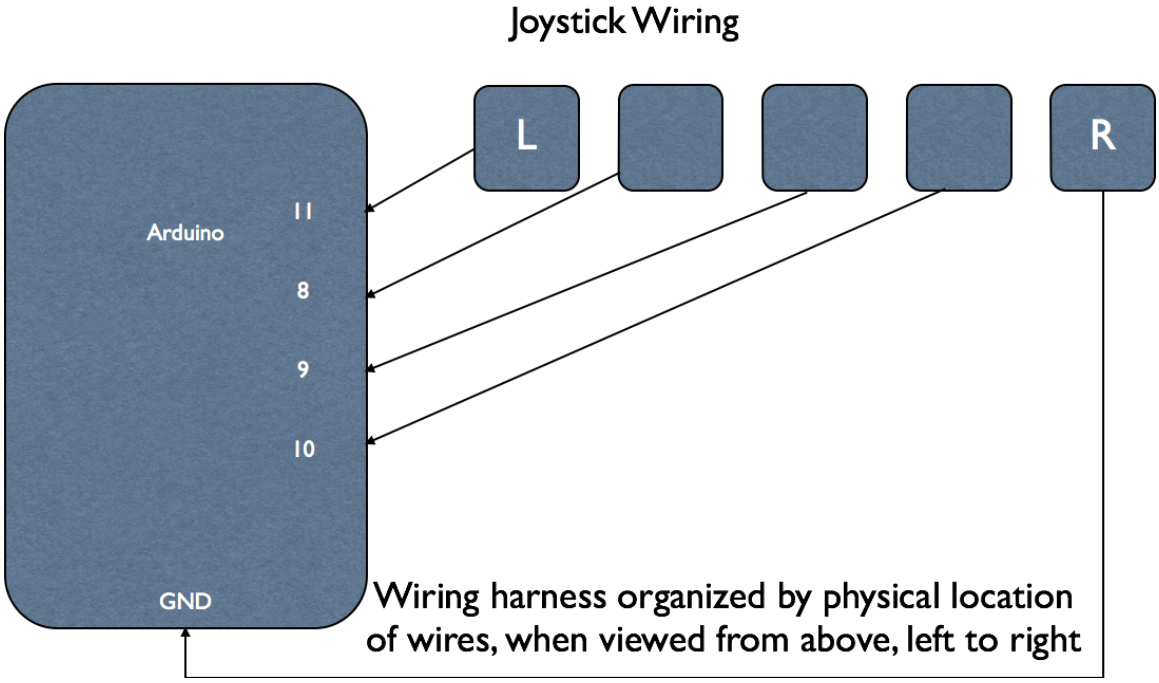

Figure 9 - Joystick Wiring Diagram

Once the buttons, Bluetooth module, and joystick are attached to the Arduino, the system is completely wired up.

Project Housing

The project housing is made from ABS plastic. It was purchased from Polycase.com an online retailer that specializes in electronics enclosures. Figure 10 shows the exact model used for this project, the LP70F. This model was chosen for two main reasons. First, it was large enough to mount all of the components using two enclosures. Second, the plastic flanges on the side of the enclosure would make it possible to use screws to mount the enclosure on a desk, table, or other surface.

Figure 10 - LP 70F Plastic Enclosure

3D Printing was also considered as an option for making the enclosures. However, purchasing a pre-made enclosure was significantly cheaper than 3D printing through a third party.

Custom templates can be uploaded with a purchase on Polycase, and the company will machine the enclosures according to custom requests. This seems like the best option for this project, however it was not available at the time this project was being completed. Instead, a drill press was used to drill holes into the plastic.

The Sega *Astro-City* arcade cabinet was chosen for the basic layout. This layout is shown in Figure 11. Instead of using an eight-button layout, the rightmost column was removed, resulting in a six-button layout. The joystick and two 24mm pushbuttons were mounted in one enclosure. The six 30mm pushbuttons were mounted in a separate enclosure. This layout was chosen because it is very ergonomic and comfortable. The buttons are staggered in a way that places them in line with each finger.

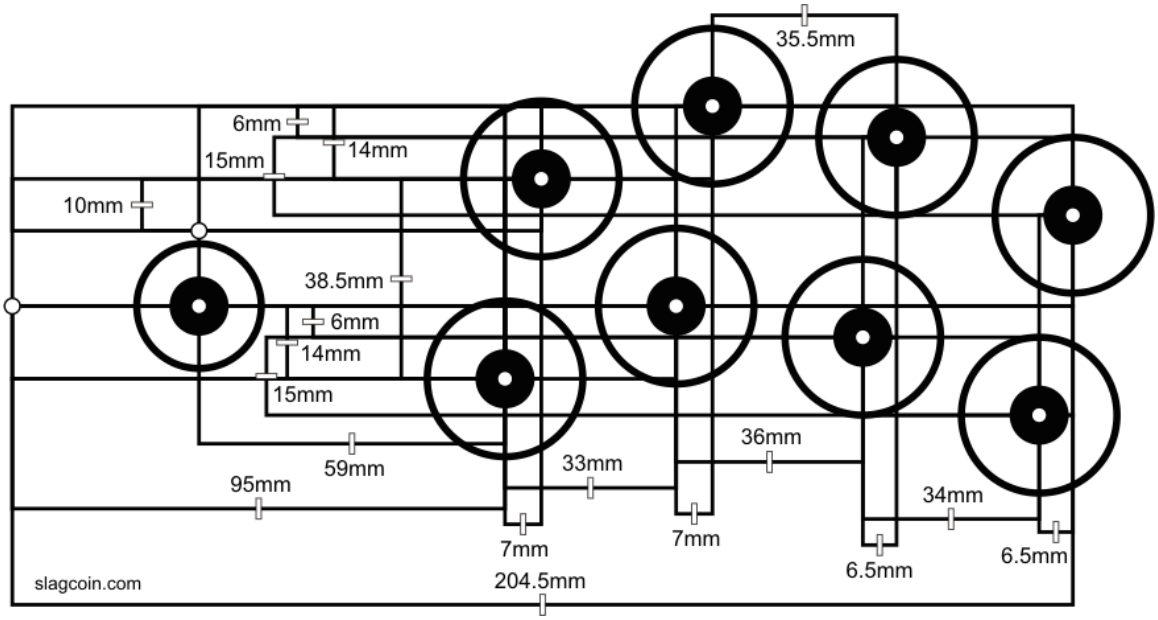

Figure 11 - Sega Astro-City Layout [3]

The drill press seemed like the most viable option, as it would require very little expertise, and it is far more available to use than a 3D printer. If a drill press is not available, a normal drill with a 30mm bit and a 24mm bit will work. The mounting hole for the joystick is also 24mm.

Figures 12 and 13 show what the buttons, microcontroller, Bluetooth modem, and joystick look like inside their housings once they have been mounted and wired properly.

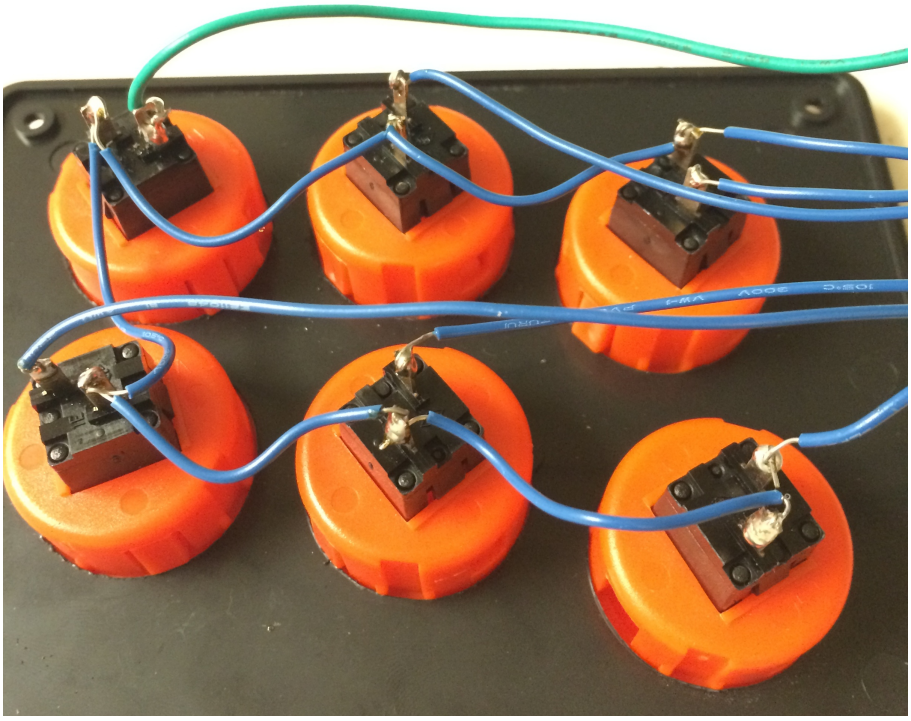

Figure 12 - 30mm buttons mounted and wired

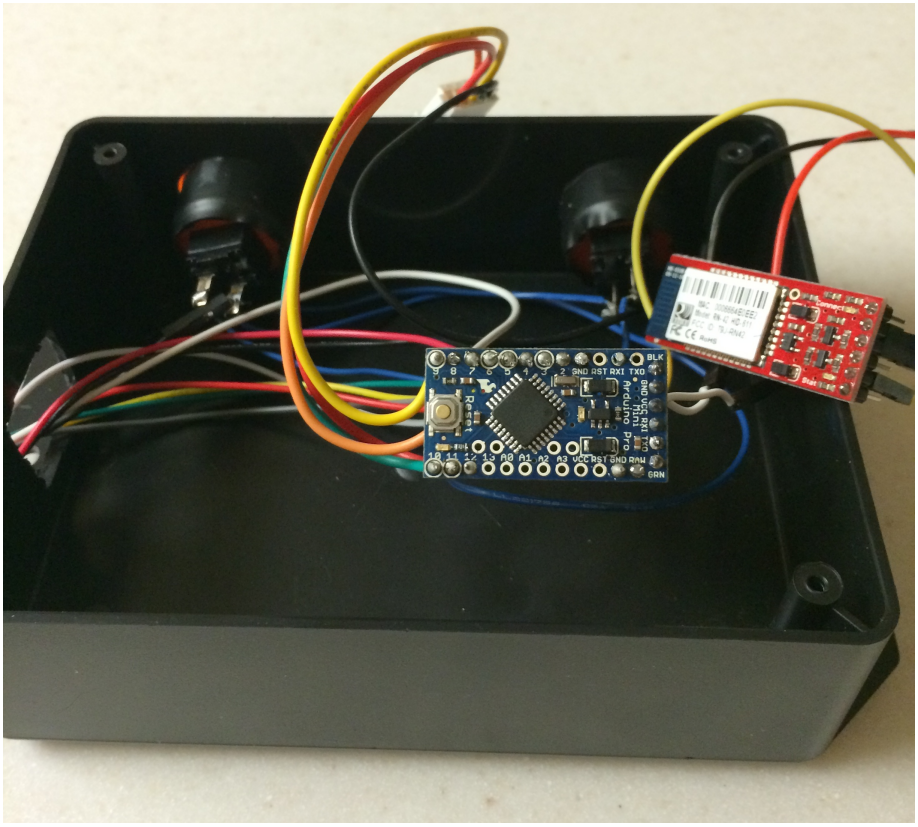

Figure 13 – Main system components: mounted and wired

Plastic was removed from part of each enclosure in order to run wire from the button's enclosure to the microcontroller's enclosure. The inside of the button enclosure can be seen in Figure 14.

Figure 14 - Enclosure with removed plastic

The final product had the male end of male-female jumpers soldered to the microcontroller. The female end of the jumper was placed outside of the enclosure, and the buttons enclosure was wired to these jumpers. This design seemed the most logical, as some games only require two buttons and a joystick. For other games, an eight-button layout isn't enough, so a new program could be loaded on the microcontroller and a different enclosure with a new button layout could be substituted in place of the current six-button enclosure.

Once the components were wired together and mounted in the enclosure, the software was programmed onto the microcontroller and the complete

system was ready for testing.

Testing

The system was tested to meet all of the criteria listed in the "Requirements," section.

Reprogrammable

The system was programmed using a reprogrammable microcontroller. I tested this by loading multiple different programs on the microcontroller.

Wireless with at least 10 ft. operating range

The programmed controller was synced to a laptop wirelessly. Proper controller function was tested from a range of 3 ft. then from 5 ft. then from 10 ft. Finally, the controller was moved 50 ft. away from the laptop and it still functioned properly.

Less than $150 parts cost

The cost of parts is outlined and explained on page 5. The controller prototype cost approximately $190, but much of these costs could easily be reduced.

Available to produce using readily available tools and equipment

The controller was made entirely with parts purchased from the internet. I had to manufacture it myself using a soldering iron, a drill press, and a screwdriver. However, all of these tools can be purchased by any individual.

Battery Powered

The controller operated from a 9V battery during testing.

Modular

The controller was designed in two modules, a joystick module and a button module. Adding and removing the button module from the joystick module tested the modularity. The controller worked with and without the button module attached.

Mac and Windows Compatible

I connected to the controller and tested the intended functionality using a Mac and using a Windows laptop.

Device automatically detected as wireless keyboard.

From the Mac and Windows laptop, the device is detected and recognized as a wireless keyboard.

Future Expandability

Because the system is composed of modules, future expandability is dependent on the same criteria as the modularity. Also, there are unused pins on the microcontroller, which can be used for future expansion as well.

Less than 16 ms. latency between controller sending a command and the operating system receiving it.

16 ms. is the time it takes for one frame to pass in a video game running at 60 frames per second. If the controller has more than 16 ms. latency, movement in the game will not occur on until the next frame is being drawn. The latency requirements can be tested heuristically. 16 ms. is slight, but still enough to be detected by the player as they input the commands.

The microcontroller is configured to be detectable as a Bluetooth keyboard. Upon powering the system, the Bluetooth module should be detectable by a computer. Its default name is “Firefly-XXXX” with the XXXX being a string of characters. From here, the “keyboard” can be connected to. For some computers, the Bluetooth module may request a password before pairing. This password is typically “1234” but otherwise will be listed in the manufacturer’s datasheet. Figure 15 shows how the Bluetooth Modem appears on a Mac.

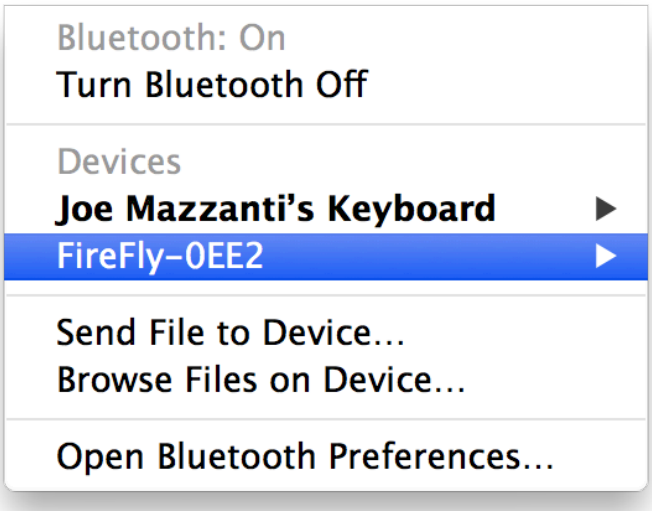

Figure 15 - Default Name for Bluetooth Keyboard

Once the Bluetooth modem pairs with the computer, proper functionality can be tested using a Text Editor program such as Notepad (for Windows) or TextEdit (for Mac). Moving the joystick or pressing buttons should produce visible feedback in the text editor program, as seen in Figure 16.

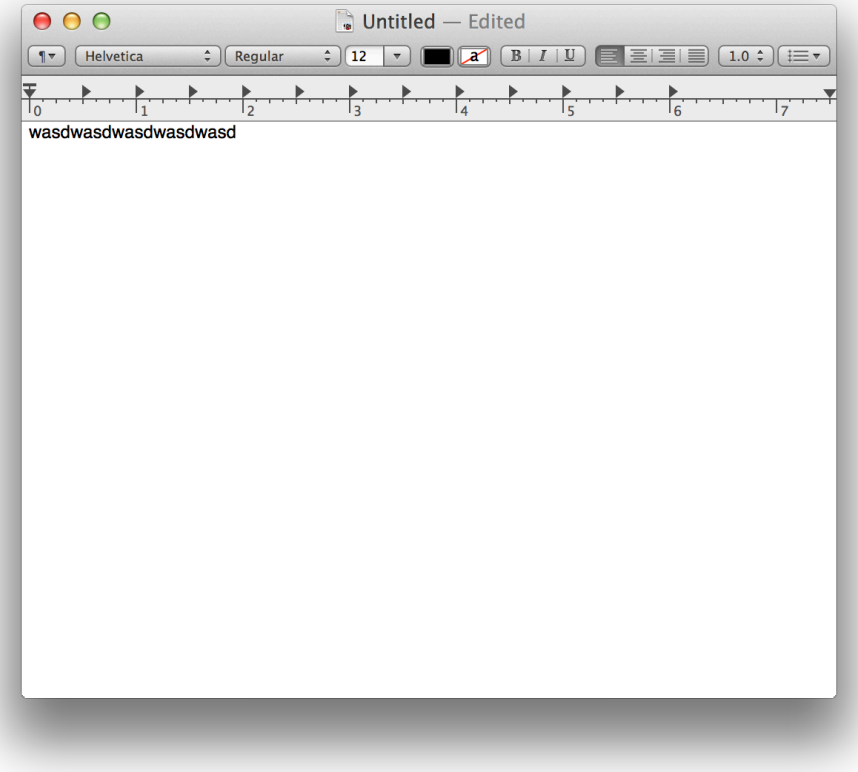

Figure 16 - System Testing using Text Editor

Once the keyboard functionality is verified, further testing was done using a video game emulator program called OpenEmu [5]. Gaming is the reason this project was developed, so this test should prove whether or not the final product works well enough for its intended purpose.

Most emulator programs have software button mapping. A keyboard press can be mapped to a virtual controller inside the game. OpenEmu supports this functionality. Figure 17 shows the microcontroller as it is mapped to the virtual controller.

Figure 17 - Software Button Mapping

The button mapping works exactly as intended. The final test is to enter a game and play the game using the Bluetooth controller. If the Bluetooth controller works properly, the game will be playable and responsive, even if the controller is moved far away from the laptop. In tests, the range of the controller was measured to be more than 50 ft. with a clear line of sight to the host laptop. Most of the tests for this system are heuristically determined. Mainly, the debounce time can be tuned through testing. Otherwise, if the controller did not work properly, it would be blatantly obvious to the user, as button inputs would be dropped, or transmission latency would be a problem. Both of these issues are incredibly noticeable, and would be grounds to improve the software program.

In practice, the system consumes about 25 mA when operating. With a standard 9v battery, rated at 565-mAh, there is about 22 hours of runtime for the system.

Conclusion and Future Work

The complete system works very well. It turned out much pricier than I

wanted. I was hoping to complete the system for less than $50. Since I already had much of the hardware required for this project, I did not end up spending much money. However, at around $100, I do not think this project exactly accomplishes what it was originally created for.

For the future, I will continue to look for areas to reduce the cost of the system. Also, other human interface device functionality will be added to the system. For instance, the joystick and two buttons could easily replace a standard two-button mouse. The joystick could control the X-Y coordinates of the mouse, while the pushbuttons could control the left and right mouse clicks.

Since the microcontroller is reprogrammable, future work could involve adding functionality of any type of human interface device. Each operating mode could be selecting by moving the joystick while the system initializes.

The housing works well and other than the loose wiring, looks good too. In the future, I would like to desolder the wiring from the microcontroller and then rewire the buttons to the system. I think it would be very wise to use an RJ-45 Ethernet cable to carry all of the button signals to the microcontroller. If not that, then a DE-9 or DA-15 connector would also simplify the wiring. It would also look much better and provide a much more secure connection.

Overall, the final product is a good start to a potentially infinitely large project. The system currently connects to a computer and acts as a wireless keyboard. It has no detectable latency, and it functions from a long distance away.

My original ideas for the project were too elaborate, and not really possible to complete in one year, by myself. With more time or more manpower, the initially outlined project may have been more feasible. Instead, the final product seems more indicative of what an individual will be able to produce with limited resources, just by following this project report and its linked resources.

Bibliography

[1] Roving Networks. “Bluetooth Data Module Command Reference & Advanced Information User’s Guide ” March 2013
Available:http://ww1.microchip.com/downloads/en/DeviceDoc/bluetooth_cr_UG-v1.0r.pdf

[2] Slagcoin Inc. “Joystick Controllers – Panel Layout” September 2008
Available: http://www.slagcoin.com/joystick/layout.html

[3] Arduino. “Arduino- Arduino Pro Mini” May 2014
Available: http://arduino.cc/en/Guide/ArduinoProMini

[4] OpenEmu “OpenEmu – Multiple Video Game System” June 2014
Available: http://openemu.org

Appendix A: Senior Project Analysis

ABET SENIOR PROJECT ANALYSIS
Human-Computer Interface for Movement Impaired Individuals

Joe Mazzanti

Dr. Bridget Benson

1. **Summary of Functional Requirements:**
    a. The Human-Computer interface serves handicap-affected individuals. Special hardware gives quadriplegics computer game control. My hardware costs $190. Comparable hardware costs $1,000. My hardware wirelessly interacts with Mac or Windows systems, registering as an input device. The input device allows control over a computer running game emulation programs such as MAME (Multiple Arcade Machine Emulation).

2. **Primary Constraints**
    a. The Human-Computer interface is mainly constrained by its form requirements. Ergonomic design will influence the form factor of the device. Special design considerations must be made regarding customer needs. The most important customer need is the customer's limited range of movement and their dexterity. Most project design decisions were affected by the form factor and ergonomic design of the device.

3. **Economic**
    a. Economic impacts exist on many levels. Human capital is consumed during the research and development of the device. Also, the device is intended to directly replace existing, expensive solutions for accessibility hardware. Replacing the existing solutions will reduce demand for said solutions, ultimately reducing the amount of jobs required for accessibility hardware designers.
    b. Producing the Human-Computer interface positively impacts financial capital; specifically, lowering the hardware costs provides an increase of money saved by the consumer.
    c. Natural resources are impacted negatively by my device's production. The device is mainly composed of plastics, whose production and recycling directly consume natural resources and pollute the environment.
    d. The original estimated cost of components is $190. The cost of labor is estimated to be approximately $1,500. Development costs incurred are approximately $700 - $2,000, based on the price of a computer for development. Because I already own the components and the development

platform, these costs have been absorbed. I will also absorb labor costs. After research and development, I will look for methods to lower the price of hardware. My project should cost less than $100 to reproduce. The estimated component costs are listed in the following table.

Table 2 - Project Cost Estimate

Part	Description	Quantity	Cost	Total
Arduino Pro Mini 328	5V, 16 MHz	1	$9.95	$9.95
BlueSMiRF HID	Bluetooth Modem	1	$44.95	$44.95
Breakaway Headers	Set of Metal Headers	1	$2.95	$2.95
Plastic Enclosure	5.50 x 4.25 x 1.75 in.	2	$4.34	$8.68
Bag of Enclosure Screws	#4-24 x 1/4" Philips	1	$3.50	$3.50
30 mm Pushbutton	Sanwa OBSF-30	6	$2.75	$16.50
24 mm Pushbutton	Sanwa OBSF-24	2	$2.25	$4.50
Leaded Solder	60/40, 0.8 mm	1	$9.95	$9.95
8 way directional joystick	Sanwa JLF	1	$23.95	$23.95
Stranded Wire	24 AWG	1	$5.97	$5.97
Soldering Iron	40 W	1	$42.39	$42.39
9V Battery		1	$2.62	$2.62
AVR Programmer	ISP for AVR	1	$14.95	$14.95
9v Battery Snap Connector		1	$1.25	$1.25
Total Cost				$192.11

e. The project is estimated to take approximately 150 hours of labor. This incorporates two iterations of design, two iterations or build, and two iterations of test. The project timeline can be expanded to incorporate additional design, build, and test iterations as determined necessary. The projected project life cycle is shown in the following Gantt chart.

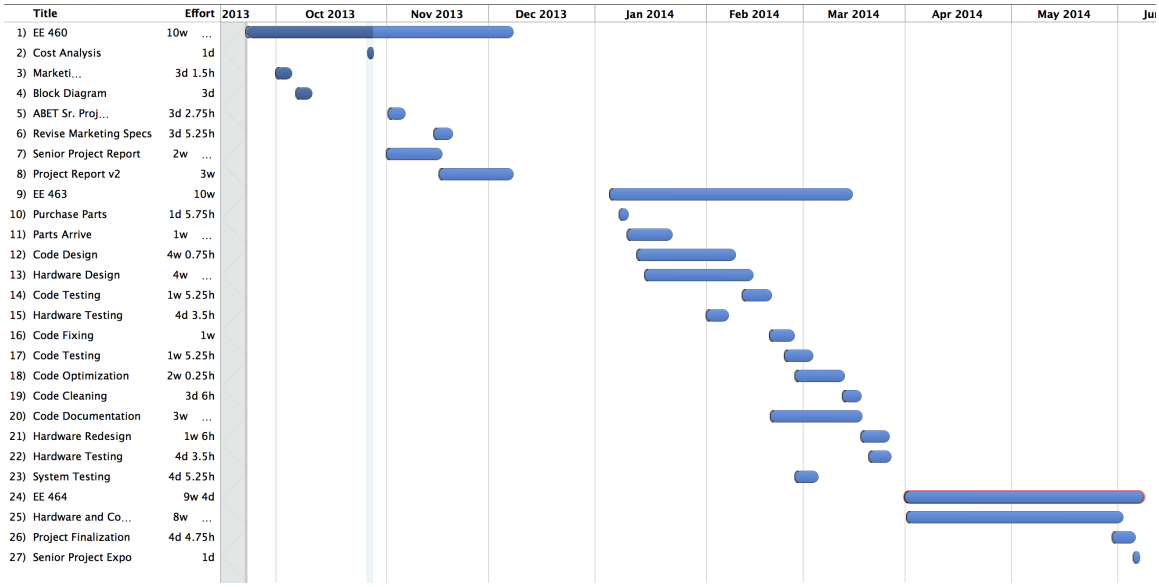

Figure 18 - Project Gantt Chart

4. **If manufactured on a commercial basis**
    a. The hardware will be made open source, therefore I plan on selling zero units in a year.
    b. The project goal is to produce hardware that can be assembled using cheap, household goods. Therefore, the manufacturing costs will be targeted towards $100 per unit. The consumer will pay these costs individually
    c. I will not make any profit from the production of this hardware.
    d. The device costs the user the initial setup cost of approximately $100, divided by the amount of time they use the device. After the initial setup is incurred, the main cost is determined by the amount the device is used. The whole system should operate using less than 100mW. If the device is used for eight hours each day, the total cost of electricity totals about $20 annually, at a rate of $0.085 / kilowatt-hour.

5. **Environmental**
    a. The environment is directly affected by my device's existence. My device relies on 9V batteries in order to operate. These batteries must be produced, polluting the environment. Once expended, the batteries must be recycled. If improperly recycled, the batteries will contribute additional pollution. The device housing will be made from silicone or some types of plastic. The manufacturing of plastics will create additional pollution, and the recycling of these plastics produces other types of waste.
    b. In order to minimize environmental impact, the device design attempts to incorporate items that a customer may already have in their home. This

increases the customer's monetary savings, increases the device's reparability and recyclability, and reduces the pollution incurred by increasing the demand for a material.

c. My device may generate more demand for petroleum-based plastics. Plastics comprise a large portion of my project's housing, and plastics

6. Manufacturability

a. Third parties currently manufacture the device's raw materials. The main manufacturing difficulty arises from the customer's ability to assemble the components into the device. I will make my design schematics and instructions publicly available, but it is up to the customer to find someone that can assemble the components into a functioning unit. Due to the specific needs of the customer, it is unlikely the customer will be able to assemble the device without additional assistance.

b. Customers with limited or no arm and leg movement are completely helpless to assemble this device. The assembly process requires hand tools such as a rotary tool and a file. These tools are difficult to use, and the lack of mobility compounds the customers' manufacturing difficulty.

c. I am making my schematics and code open source, under the assumption that a caretaker, friend, or family member will perform the manufacturing and assembly of a complete device. The largest assembly difficulties they face will directly relate to their experience working with power tools, or microcontrollers.

d. No difficulties should arise while acquiring device components. The entire device is designed with readily available parts, which can be found online or in many non-specialty retail stores.

7. Sustainability

a. The maintenance and sustainability of the completed device will vary. The customers' ability to produce and maintain a working device is limited to their access to materials and their ability to have the device assembled. The project is very reliant on software, making the initial upgradability trivial. Keyboard macros and additional platform support can be added through firmware updates. The hardware capabilities limit the extent of software upgradeability.

b. The customer can upgrade the hardware. The customer may add additional input buttons. The customer may support additional microcontroller platforms through source code modification. I will not support many microcontrollers initially. The source code will be written in C, and the code is highly portable.

c. I will continue updating the device schematics as criticism and feedback are received online. Steadily improving the device's features increases the overall sustainability of the project. Also, design flaws will become apparent through user testing and feedback, which will not be possible until after my project is completed and published online.

8. **Ethical**
    a. This device is designed from the principle of the Golden Rule. If I had a disability, I would want affordable hardware that can improve the quality of my life. Cheaper accessibility hardware will lower the price of other hardware on the market. Lower market prices may affect the demand for hardware jobs. I am willing to sacrifice a few jobs to increase the quality of life for a large disabled community.
    b. My project directly complies with the IEEE code of ethics [11]. My device is designed to serve all people. Item 8 of the IEEE code of ethics states that IEEE members are committed to treat all person fairly, regardless of disability. My device is specifically designed to accommodate disabilities. This also adds information to the existing body of work, improving future accessibility products and their design process.
    c. In compliance with IEEE section 7.8 items 5 and 6, it is my duty to "improve the understanding of technology, its appropriate application, and potential consequences." It is also my duty "to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations." My project complies with both of these items. First, my device improves the understanding of human computer interaction. Second, my project serves as the capstone of my electrical engineering education. My education taught me the skills required to complete this project. Instead of relying on hobbyists to create this hardware, I believe it is my duty to undertake this task for the purposes of safety.

9. **Health and Safety**
    a. Potential health concerns have been considered. First, there is a small risk of electric shock if the device is improperly assembled. The device may be in contact with water and saliva. Proper waterproofing of the device will circumvent electric shock risks associated with the device.
    b. 9V batteries power the device. These batteries are safe to interact with personally. No other external power sources exist.
    c. Smaller parts of the device are small enough to be swallowed. The parts cannot be swallowed if they are properly contained within the device housing.
    d. Similar devices contain wires that tether the user to their computer. This poses as a potential choking hazard, or a fire hazard due to reduced mobility. My device is wireless, improving user mobility, and preventing accidents that may occur due to wires.

10. **Social and Political**
    a. The project directly impacts individuals affected with disabilities. The direct stakeholders are those who will ultimately use the product. Current accessibility engineers are indirect stakeholders. Caretakers for the disabled are also direct stakeholders.

b. The project directly aids the hardware community, as the current pool of knowledge is enhanced with new information, the quality of accessibility hardware will increase.
c. The project will affect indirect stakeholders both positively and negatively. Saturating the accessibility market with cheaper hardware will reduce the demand for more expensive hardware, ultimately affecting the price of alternative solutions. This will affect the accessibility industry's operation in a negative way, as lowering prices will lower profits. However, the positive effect is that my project will bring greater visibility to their products, generating additional revenue for their company.

11. Development

a. During the course of my project, I learned how to incorporate the Bluetooth wireless communication protocol into my microcontroller designs.
b. I learned how to use a microcontroller as a human interface device.
c. I learned how to minimize microcontroller power consumption in order to maximize battery life.
d. I learned how to machine plastic using a drill press and rotary tool.
e. I learned how the HID protocol works, and how to program HID compliant devices.

Appendix B: Source Code

```
// test code for sending keystrokes from Arduino
// to computer via HID Bluetooth module
// This version has the Arduino and Bluetooth module acting like a
wireless keyboard.

#include <avr/io.h>

uint8_t upInput = 8; //Pin numbers for the Cardinal Directions
uint8_t leftInput = 9;
uint8_t rightInput = 10;
uint8_t downInput = 11;

#define upKey 0x01   //Used for Bit Masking each direction
#define leftKey 0x02
#define rightKey 0x04
#define downKey 0x08

uint8_t button1Input = 2; //Pin numbers corresponding to each button
uint8_t button2Input = 3;
uint8_t button3Input = 4;
uint8_t button4Input = 5;
uint8_t button5Input = 6;
uint8_t button6Input = 7;
```

```
uint8_t button7Input = 12;
uint8_t button8Input = 0;

//Bit Masks for each button, along with HID scan code values
#define button1 0x04 //Y = 0x1C; T = 0x17;
#define button2 0x08 //U = 0x18; F = 0x09;
#define button3 0x10 //I = 0x0C; M = 0x10;
#define button4 0x20 //O = 0x12; G = 0x0A;
#define button5 0x40 //H = 0x0B; R = 0x15;
#define button6 0x80 //J = 0x0D; N = 0x11;
#define button7 0x10 //K = 0x0E; P = 0x13;
#define button8 0x01 //L = 0x0F; V = 0x19;

//LED pin used for debugging. It should not be used as a button input.
uint8_t led = 13;

//Variables tracking which buttons are pressed and which direction the
stick is being held in
uint8_t stickState = 0;
uint8_t buttonState = 0;

//Arrays tracking which buttons are currently active
uint8_t activeKeys[6] = {
  0};
uint8_t listOfPortDButtons[7] = {
  button8, button1, button2, button3, button4, button5, button6};
uint8_t listOfPortBButtons[5] = {
  upKey, leftKey, rightKey, downKey, button7};
uint8_t numberOfBButtons = 5;
uint8_t numberOfDButtons = 7;

//DirectionTransmissions
uint8_t keyPress = 0x00;
uint8_t keyTwoPress = 0x00;
uint8_t upRelease = 0x00;
uint8_t leftRelease = 0x00;
uint8_t rightRelease = 0x00;
uint8_t downRelease = 0x00;

//Initialize the button states to 0
uint8_t button1Press = 0x00;
uint8_t button2Press = 0x00;
uint8_t button3Press = 0x00;
uint8_t button4Press = 0x00;
uint8_t button5Press = 0x00;
uint8_t button6Press = 0x00;
uint8_t button7Press = 0x00;
uint8_t button8Press = 0x00;

uint8_t button1Release = 0x00;
uint8_t button2Release = 0x00;
uint8_t button3Release = 0x00;
uint8_t button4Release = 0x00;
uint8_t button5Release = 0x00;
uint8_t button6Release = 0x00;
uint8_t button7Release = 0x00;
uint8_t button8Release = 0x00;
```

```
int8_t xValue = 0;
int8_t yValue = 0;
int8_t zValue = 0;
int8_t rotValue = 0;


void setup(){

  //set input states
  pinMode(upInput, INPUT);
  pinMode(downInput, INPUT);
  pinMode(rightInput, INPUT);
  pinMode(leftInput, INPUT);

  pinMode(button1Input, INPUT);
  pinMode(button2Input, INPUT);
  pinMode(button3Input, INPUT);
  pinMode(button4Input, INPUT);
  pinMode(button5Input, INPUT);
  pinMode(button6Input, INPUT);
  pinMode(button7Input, INPUT);
  pinMode(button8Input, INPUT);

  pinMode(led, OUTPUT);

  //set pull-up resistors
  digitalWrite(upInput, HIGH);
  digitalWrite(downInput, HIGH);
  digitalWrite(leftInput, HIGH);
  digitalWrite(rightInput, HIGH);

  digitalWrite(button1Input, HIGH);
  digitalWrite(button2Input, HIGH);
  digitalWrite(button3Input, HIGH);
  digitalWrite(button4Input, HIGH);
  digitalWrite(button5Input, HIGH);
  digitalWrite(button6Input, HIGH);
  digitalWrite(button7Input, HIGH);
  digitalWrite(button8Input, HIGH);

  digitalWrite(led,LOW);

//the joystick switches are only present on PORTB
//the buttons are used in PORTD, but PIN1 of portD has no button
//The button state variable is bit masked to account for this

  stickState = PINB;
  buttonState = PIND & 0xFD;


  Serial.begin(250000); //begin serial communication at 250000 baud (0%
clock error using 16 MHz clock)
}

void loop()
{
```

```
    if( (PINB != stickState) || ((PIND & 0xFD) != buttonState) )
    {
      delay(4);
      stickState = PINB;
      buttonState = PIND & 0xFD;
      updateActiveKeys();
    }
}

//////////////////////////////////////////////////////////////////////

uint8_t getState(uint8_t REG, uint8_t KEY)
{
  return (REG & KEY) ^ KEY;
}

//////////////////////////////////////////////////////////////////////

void updateActiveKeys(){
  // release unpressed keys
  for (uint8_t i = 0; i < 6; i++) {
    if (activeKeys[i]) {
      uint8_t key = getKeyMask(activeKeys[i]);
      uint8_t port = getPortForKeyValue(activeKeys[i]);
     //if the key is not being pressed, it's spot in the HID report is
replaced with a release value of 0x00.
      if(!getState(port, key)) {
        activeKeys[i] = 0x00;
      }
    }
  }
  // send release message
  sendButtons();

  // add newly pressed keys
  for (uint8_t i = 0; i < numberOfDButtons; i++) {
    if(getState(buttonState, listOfPortDButtons[i])){
      uint8_t keyValue = getPortDKeyValue(listOfPortDButtons[i]);
      uint8_t firstEmptySlot = 10;
      uint8_t found = 0;
      for (uint8_t j = 0; j < 6; j++) {
        if ( (activeKeys[j] == 0x00) && (firstEmptySlot > 5) ) {
          firstEmptySlot = j;
        }
        else if (activeKeys[j] == keyValue) {
          found = 1;
          break;
        }
      }
      if (!found && firstEmptySlot < 6) {
        activeKeys[firstEmptySlot] = keyValue;
      }
    }
  }

  for (uint8_t i = 0; i < numberOfBButtons; i++) {
```

```
    if(getState(stickState, listOfPortBButtons[i])){
      uint8_t keyValue = getPortBKeyValue(listOfPortBButtons[i]);
      uint8_t firstEmptySlot = 10;
      uint8_t found = 0;
      for (uint8_t j = 0; j < 6; j++) {
        if ( (activeKeys[j] == 0x00) && (firstEmptySlot > 5) ) {
          firstEmptySlot = j;
        }
        else if (activeKeys[j] == keyValue) {
          found = 1;
          break;
        }
      }
      if (!found && firstEmptySlot < 6) {
        activeKeys[firstEmptySlot] = keyValue;
      }
    }
  }

  // send new HID report with buttons added
  sendButtons();
}

//////////////////////////////////////////////////////////////////////

uint8_t getPortBKeyValue(uint8_t KEY){

  uint8_t value;

  switch (KEY) {
  case upKey :
    value = 0x1A;
    break;
  case downKey :
    value = 0x16;
    break;
  case leftKey :
    value = 0x04;
    break;
  case rightKey :
    value = 0x07;
    break;
  case button7 :
    value = 0x0E;
    break;
  }

  return value;
}

//////////////////////////////////////////////////////////////////////

uint8_t getKeyMask(uint8_t VALUE){

  uint8_t mask;

  switch (VALUE) {
```

```
  case 0x1A :
    mask = upKey;
    break;
  case 0x16 :
    mask = downKey;
    break;
  case 0x04 :
    mask = leftKey;
    break;
  case 0x07 :
    mask = rightKey;
    break;
  case 0x0E :
    mask = button7;
    break;
  case 0x1C:
    mask = button1;
    break;
  case 0x18:
    mask = button2;
    break;
  case 0x0C :
    mask = button3;
    break;
  case 0x12 :
    mask = button4;
    break;
  case 0x0B :
    mask = button5;
    break;
  case 0x0D :
    mask = button6;
    break;
  case 0x0F :
    mask = button8;
    break;
  }

  return mask;
}

////////////////////////////////////////////////////////////////////////

uint8_t getPortDKeyValue(uint8_t KEY){

  uint8_t value;

  switch (KEY) {
  case button1:
    value = 0x1C;
    break;
  case button2:
    value = 0x18;
    break;
  case button3 :
    value = 0x0C;
    break;
```

```
  case button4 :
    value = 0x12;
    break;
  case button5 :
    value = 0x0B;
    break;
  case button6 :
    value = 0x0D;
    break;
  case button8 :
    value = 0x0F;
    break;
  }

  return value;
}

//////////////////////////////////////////////////////////////////////


uint8_t getPortForKeyValue(uint8_t VALUE) {

  switch (VALUE) {
  case 0x1A:
  case 0x16:
  case 0x04:
  case 0x07:
  case 0x0E:
    return stickState;
    break;
  case 0x1C:
  case 0x18:
  case 0x0C:
  case 0x12:
  case 0x0B:
  case 0x0D:
  case 0x0F:
    return buttonState;
    break;
  }
}



//////////////////////////////////////////////////////////////////////


void sendButtons() {
  Serial.write(0xFD);
  Serial.write(0x09);
  Serial.write(0x01);
  Serial.write(0x00);
  Serial.write(0x00);
  for (uint8_t i = 0; i < 6; i++) {
    Serial.write(activeKeys[i]);
  }
}
```

Appendix C: User Manual

1. Once the system is completely assembled, wired, and programmed, power the system with a 9v battery.
2. Once the system is powered, a red light should be visible on the Bluetooth modem. Using a Mac or Windows computer, follow the steps required to add a Bluetooth keyboard. For Windows, follow this guide (http://windows.microsoft.com/en-us/windows7/add-a-bluetooth-enabled-device-to-your-computer)

   On a Mac, follow this guide (http://support.apple.com/kb/PH13704)

3. Once the device is connected to the computer, it should automatically function as a Bluetooth keyboard. Test to make sure the keyboard functionality is working properly by going opening a program that uses keyboard inputs. A text editor is good for this purpose.
4. Once in the program, move the joystick and press the buttons to see if the joystick movement and button presses are converted to key presses.
5. If the joystick and buttons are working properly, the controller is working properly, and can be used within the emulator program. If the joystick and buttons do not work properly, check the solder connections and ensure there is no solder bridging the pads of the microcontroller. If there are no issues with the soldering, reprogram the microcontroller and verify that the proper program is being uploaded, and that the upload finishes without error.
6. If the controller still does not work, check the buttons and joystick using a continuity tester to see that the switches work properly. The aforementioned issues will comprise the majority of possible system errors.

This concludes the user manual.